Open Source Software: Opportunities and Challenges

David Tuma
Software Process Dashboard Initiative

Much more than a buzzword, open source software is becoming an increasingly important part of the information technology environment. Many program managers, project managers, and developers in the Department of Defense and elsewhere are already familiar with open source; others may wonder how best to use open source in a project environment. In considering the opportunities presented by open source software, it is helpful to understand its origins as well as the challenges you may face in implementing it.

Whether you realize it or not, you rely on open source software every day. Open source software provides the underpinnings for the Internet, directs much of the world's e-mail traffic, and powers more than two-thirds of the world's Web sites [1].

The open source paradigm is based on the idea that software reuse need not stop at organizational boundaries. By sharing source code freely under a license that generously permits copying, modification, and redistribution, open source projects allow collaborative software development that benefits a larger community.

Although open source software has been in existence for decades, the arrival of the Internet has led to a veritable explosion in open source activity. The Internet has made it possible for developers around the world to discover each other, collaborate in real time, and share the works they create with others.

Organizations, businesses, and governments around the world are opening up to the possibilities provided by open source software. Although open source software has been in use for some time in the Department of Defense (DoD), a policy released in May of 2003 [2] officially put open source software on a level playing field with proprietary software.

If you have not explored open source software firsthand, you might be surprised by its diversity. Although media coverage commonly focuses on the Linux operating system and on server-based applications like Apache and MySQL, these applications are only the tip of the iceberg. System administration tools like Snort (an intrusion-detection tool), Nessus (a vulnerability scanner), and NetCat (a network debugging and mapping tool) help to manage computer networks. A wide variety of tools ranging from the Gnu's Not Unix (GNU) C Compiler to Perl (a popular programming language) to Eclipse (a Java integrated data environment developed by IBM) target software development. An extensive array of reusable libraries and frameworks can save your software project time and money. Applications like Mozilla (a Web browser) and OpenOffice (an office productivity suite) address common end-user needs. And of course, the list goes on – one popular open source Web site alone lists more than 11,000 stable/mature open source projects [3].

Strengths of Open Source

Many organizations are initially attracted to open source products because they can generally be acquired for free. Momentarily deferring the debate about total cost of ownership until later in this article, removal of the initial procurement barrier can at times be a significant enabling factor for software use. For example, budget constraints have encouraged academia to adopt and support open source software for decades. With the increased public awareness of open source, public and private middle and secondary schools are beginning to investigate open source options, as well [4]. And small businesses may find open source software helpful in *leveling the playing field.*

Government organizations and contractors often discover different reasons to appreciate zero-cost licenses for open source software. For example, in 1999 the Census Bureau needed to create a Web site but had no official information technology budget to make it happen; staffers were able to build the Web site using open source applications and existing hardware, and the Web site is still in use today [5]. In addition, open source approaches can lower the monetary risk of experimenting with new technologies, effectively speeding the pace of technology adoption and supporting the collaborative development of new standards [1]. Moreover, the simplified licensing model of open source software can facilitate inter-agency sharing and reuse of developed solutions [6].

In fact, reuse is a central tenet of the open source development paradigm. Open source software licenses (as defined by the Open Source Foundation [7]) explicitly target software reuse by permitting source code to be copied, modified, and distributed. When organizations and individuals share a common need, they can share and reuse entire open source products rather than independently developing redundant code. Common examples of this style of reuse include not only off-the-shelf open source products like the Apache Web server, but also application frameworks like Struts, and reusable libraries for performing tasks such as parsing eXtensible Markup Language or consuming Web services. Communities organized around this type of open-source reuse foster rapid innovation and progress, as many contributors can simultaneously develop improvements that benefit all users of the product.

The true strength of open source reusability, however, emerges when an organization or an individual has a unique need. An organization or individual with a new or unmet need is free to modify an open source product[1] to meet that need, potentially reusing thousands of lines of code. The benefits of reuse in such a scenario are well understood, saving countless hours of development, testing, and maintenance. Contributing such an enhancement back to the open source community can benefit other organizations with similar needs [8].

Open source software promotes reuse in another, unexpected way through code transparency. Inadequate documentation has long been identified as a significant barrier to software reuse; in addition, subtle misunderstandings between developers on either side of a code boundary can lead to insidious errors. In the absence of flawless code and impeccable documentation, the freedom to examine source code can mean the difference between a useful, reusable library and a baffling *black box.*

When code in an open source library

behaves unexpectedly, a developer can peer into it to understand its operation and true intent, and determine whether the defect lies in the library or in their mental model of it. If the defect is in the library, the developer can either notify the original author or fix it himself or herself. Otherwise, the developer can gain a better understanding of the library and correct his or her code to use it properly. In either case, transparency across the code boundary helps to improve the quality of the overall system.

In the broader pursuit of software quality, many open source projects succeed by leveraging code review practices on a massive scale. Studies have demonstrated that code reviews (analyzing source code for problems) can remove defects much more effectively than testing (running an application and watching for incorrect behavior). In a closed source environment, only internal developers can perform code reviews, while the larger user community is constrained to *black-box testing*. Open source projects remove this limitation, freeing any end user to participate in the code review process.

While there is a practical limit to the number of software developers that can work together on a single team (because communication needs increase exponentially with the number of developers), there is almost no limit to the number of people who can simultaneously review code and test an application [9]. Accordingly, successful open source projects like Linux harness the skills of their large user base to perform massively parallel code reviews. Even brute-force testing methods can prove effective in improving product quality when thousands of individuals participate, each testing a product from his or her own unique perspective. When the conditions are right, open source projects can successfully employ these techniques to develop very high-quality products [10].

Security of Open Source Software

Open source proponents cite the collaborative review process as a major strength of the open source paradigm, ideally suited for producing highly reliable, secure code. Nevertheless, the security of open source software (and its comparison to the security of proprietary software) is a hotly debated topic.

For example, open source critics have recently questioned the use of Linux for defense systems. In a recent press release, Green Hills Software, Inc.'s Chief Executive Officer Dan O'Dowd stated, "The open source process violates every principle of security. It welcomes everyone to contribute to Linux. Now that foreign intelligence agencies and terrorists know that Linux is going to control our most advanced defense systems, they can use fake identities to contribute subversive software that will soon be incorporated into our most advanced defense systems" [11].

Other experts quickly responded to O'Dowd's claims, labeling them *fear, uncertainty, and doubt*. Citing Easter eggs, backdoors, spyware, and malware, they pointed out that proprietary software could just as easily contain illegitimate code. Citing the rigorous public review process used to approve Linux code, they argued that a *foreign intelligence agency* could more easily infiltrate a commercial development project than slip malevolent code under the noses of hundreds or thousands of watching reviewers. If one is truly worried about such malicious code, they argued, open source development is a better approach than closed source development since it permits anyone to perform his or her own independent review [12, 13].

> *"An organization or individual with a new or unmet need is free to modify an open source product to meet that need, potentially reusing thousands of lines of code."*

Experts on both sides of the open source security debate contribute many compelling arguments. The bottom line, however, is that open source software is not automatically more or less secure than proprietary software [14]. Both development approaches have their strengths and weaknesses, but neither automatically produces more secure code than the other. Unfortunately, impassioned people on both sides of the debate regularly make broad, unconditional assertions about the relative security of open source and proprietary software. Although such statements certainly keep the debate interesting (and make for colorful news items), objective analyses are more useful. An astute observer should be skeptical of sweeping generalities and dig deeper to find impartial expert analysis.

Using Open Source Software

Many people may have preconceived ideas about potential uses of open source software. With the variety of products available, however, there are many ways projects might consider using open source (including but not limited to the following):

- Deploy onto *off-the-shelf* open source server software (such as Linux, Apache, or MySQL).
- Reuse open source architectural frameworks (such as Struts, Spring, or Zope).
- Make use of open source development tools (such as Ant, Eclipse, or CVS [Concurrent Versions System]).
- Leverage reusable libraries (such as Xalan, OpenSSL, or GTK+).

Like the DoD, many organizations currently have policies that permit using open source software when it meets applicable requirements and provides the best value for the money. Thus, project managers considering using open source software must be prepared to analyze all the options available – both proprietary and open source – and determine which product provides the best value within the requirements for the project at hand. Project managers can immediately encounter several challenges relating to open source products.

For example, project managers may have difficulty discovering what open source products are available. Unlike proprietary alternatives, open source products rarely have budgets for advertising and marketing. And while mainstream media often includes news items on *flagship* open source products like Linux and Apache, you are unlikely to find information on frameworks, libraries, tools, or less common applications.

Fortunately, many Internet resources are available. Two of the largest Web sites are <freshmeat.net> [15] and <sourceforge.net> [3]; both list tens of thousands of open source software items in a categorized and searchable format. Keep in mind that open source (like technology in general) can progress at a remarkable rate, and new open source products and frameworks can seemingly appear overnight. Similarly, an open source project that might have had too many rough edges six months ago may now exceed your needs today. To keep abreast of these changes, software developers may find <slashdot.org> [16] to be a useful source

of product announcements.

With a list of potential open source and proprietary options ready, the daunting challenge of determining best value begins (see Table 1). This project management task has never been simple, but open source introduces many additional challenges.

For example, when considering only proprietary options, managers might glean information from the *market price* of a product. It might be a safe working assumption that a $50,000 product has more features than a $500 product. Where, then, does an open source product fit in the list? In a similar vein, managers can often look at *market share* statistics for proprietary products to see which ones are most popular. Unfortunately, this is often impossible for all but a handful of open source products. Although the Hyper Text Transfer Protocol makes it possible to estimate the number of Apache Web servers in use around the world, there is almost no way to determine how many people are using Linux, OpenOffice, or many of the tens of thousands of other open source applications in existence. These challenges make it more difficult to build a *short list* of products to choose from.

As a result, finding the *best value* comes down to a lot of research, legwork, and analysis. Find people within your organization and elsewhere who are using the products, and draw upon their experiences for pragmatic advice. Look for reviews in online publications. And above all, be wary of sweeping claims that *open source software is better/worse than proprietary software in category XYZ*. Although it is tempting to listen to such claims (because they would certainly simplify your decision-making process), they rarely withstand careful scrutiny. In truth, comparisons must be made on a case-by-case basis, taking into consideration not only the products in question but also the unique requirements of your project. Here are some items to include on your comparison checklist (see [17] for a more thorough checklist):

- **Requirements.** This is, of course, one of the most important characteristics to consider: Does the software provide the functionality your project needs? If an open source product is missing a small function you need, is it possible and cost effective to add the feature yourself (keeping life-cycle costs in mind)? Does it meet your project's requirements for performance, quality, reliability, scalability, and security?
- **Licensing Restrictions.** Open source software is distributed under various licenses with differing terms. If some of these are incompatible with your project's target environment (see discussion below), they should be eliminated early in the selection process.
- **Support.** What quality of support is available for the various options on your list, and how much does that support cost? Support for open source products may be available from the original developers or from third parties. If third-party support is not available, you can gauge the level of support from the open source community by scanning related help forums, bug trackers, and mailing list archives. Throughout the past months, have user cries for help gone unanswered, or have they been addressed quickly? Larger open source projects – especially the *flagship* open source products like Linux and Apache – often have excellent support [18, 19, 20]. Support for smaller projects may be lacking – especially if the project is no longer under active development. In such cases, you will need to estimate how much it would cost to support the application yourself.
- **Documentation.** Is the product adequately documented? Does the documentation appear up-to-date? Are third-party books on the product available?
- **Maintenance Costs.** Will the deployed product be easy to maintain? For example, will it require monitoring and patching, and are tools available to help with those tasks?
- **Skills.** Do members of your team have expertise with the products in question? If not, is training available (either online or from a third party)? If you plan to outsource or subcontract project work or maintenance, are experienced contractors/integrators available?
- **Warranties.** Open source software typically comes with no warranties, although third-party warranties may be available. How do these compare with the warranties for the proprietary software choices on your list?
- **Vendor Lock-In.** Is the product standards-based, or does it lock you into a particular proprietary solution? Although most open source products are vendor-neutral, not all are. If technology neutrality is important to your project, examine your options carefully.

Ultimately, many of these considerations roll up into the larger concept of *total cost of ownership* (TCO). TCO has received a lot of media attention lately, and will continue to be a source of debate; like all of the characteristics above, however, TCO must be evaluated on a case-by-case basis. Some projects will see a lower TCO from proprietary solutions, while some will see a lower TCO from open source products. And with certainty, the types of projects that benefit from each will change over time, as both proprietary and open source products move forward.

If your research and analysis lead you to select an open source product for your project, it is, of course, imperative that you understand and respect the license terms of that open source software[2]. Because open source software is generally available at no cost, people often mistakenly assume that the code is in the public domain and can be used without restrictions. On the contrary, open source software is generally distributed under one of the licenses approved by the Open Source Initiative [7].

By definition, open source licenses universally grant broad permission to copy, modify, and distribute source code and compiled binaries, as long as the terms of the license agreement are respected. In many cases, these terms are very simple to comply with; for example, they may require a specific copyright notice and disclaimer to be included in the end-user documentation of a product that redistributes an open source library. Of course, the terms vary from license to license, and dozens of open source licenses are in active use today, so it is important to carefully read, understand, and comply with the licenses of any open source products you use. Fortunately, this task is not as difficult as it sounds, since a small number of licenses (listed in Table 2) cover as much as 90 percent of the open source software currently available.

If you modify an open source product

Table 1: *Checklist for Comparing Software Options*

Checklist for Comparing Software Options

Cost-Related Factors

- ☐ Software Costs (purchase, upgrades, licensing)
- ☐ Hardware Costs (purchase, upgrades)
- ☐ Staffing Costs (internal and contract staff)
- ☐ Internal and External Support Costs (installation, maintenance, troubleshooting)
- ☐ Indirect Costs (downtime, training)

Qualitative Factors

- ☐ Customizability/Flexibility
- ☐ Availability/Reliability
- ☐ Interoperability
- ☐ Scalability
- ☐ Performance
- ☐ Security
- ☐ Manageability/Supportability
- ☐ Expected Lifetime

Source: "A Business Case Study of Open Source Software" [17]

or compile it into a larger program, additional licensing terms may apply. For example, some licenses will require your modifications to be released back to the open source community. Ensure that you read the license carefully and understand its requirements. In particular, most organizations will want to avoid compiling and linking code distributed under Gnu's Not Unix (GNU) General Public License into a larger project [21].

When using, modifying, or enhancing open source software, it is also important to understand any applicable restrictions that stem from organizational policy, contractual requirements, and the like. For example, if your organization forbids any external release of code, and a particular open source product requires distributed code modifications to be released as open source, then you may not be able to modify that library and still meet all your legal obligations[2]. From a project management standpoint, it is best to be aware of such constraints before heading down a dead-end path.

Participating in the Open Source Community

As awareness of open source software grows, and as open source usage becomes a more common part of everyday software development, more and more individuals and organizations wonder how they can get involved with the open source community.

Some organizations have successfully embraced the open source development model for their managed projects, accepting code contributions from external developers. However, since external developers may not be accountable to the internal project goals, this approach introduces risks that most projects are not able to accept. Fortunately, there are still many other creative ways to work with the open source community.

Perhaps the simplest way to participate in the open source community is to provide feedback and bug fixes to open source projects. If your project uses an open source product (whether it be an operating system, an application, a framework, a reusable library, or some other product), take the time to thank the developers who created it. In many cases, this thanks is the only payment they receive for their efforts. If you discover and/or fix a bug in the product, you can benefit the entire community by sharing your discovery or patch with the product developers.

Another way to participate in the open source community is to contribute enhancements to an open source product. If you discover that a particular open source product meets most (but not all) of your needs, and decide (through due diligence) that the best course of action for your project is to extend and enhance the open source product, consider contributing these enhancements back to the community when your project is done. Many commercial and government projects participate in open source in this way.

Some organizations have gone even further, contributing completed projects in their entirety to the open source community. In addition to the obvious benefits of reuse, organizations have discovered other unexpected benefits as well – for example, reduced life-cycle costs as open source developers begin fixing bugs and adding features [22]. Both government and corporate supporters of open source are increasingly using this approach.

Summary

Open source will continue to be an important part of the software landscape for years to come. Although misconceptions and misinformation often confuse the decision-making process, careful analysis can indicate where using open source is appropriate. Understanding the issues and opportunities inherent in open source is the first step in using it effectively to deliver maximum value for your project, your organization, and your clients.◆

Commonly Used Open Source Licenses
• GNU General Public License • GNU Lesser General Public License • Berkeley Software Distribution License • Artistic License • Apache Software License • Massachusetts Institute of Technology License • Mozilla Public License

Table 2: *Commonly Used Open Source Licenses*

References

1. Fordahl, Matthew. "Open-Source Software a Big Tech Player." AP Online 16 July 2004 <www.bizreport.com/news/7684>.
2. Stenbit, John P. "Open Source Software (OSS) in the Department of Defense (DoD)." Washington, D.C.: Defense Information Systems Agency, 28 May 2003 <http://iase.disa.mil/oss-in-dodmemo.pdf>.
3. Open Source Technology Group. SourceForge.net 12 Sept. 2004 <http://sourceforge.net>.
4. Surran, Michael. "Making the Switch to Open Source Software." T.H.E. Journal 31.2 (Sept. 2003): 36+ <www.thejournal.com/magazine/vault/a4499.cfm>.
5. Zieger, Anne. "Open-Minded: Government Agencies Are Overcoming Obstacles to Open Source." Government Enterprise 2 June 2002 <www.governmententerprise.com/showArticle.jhtml?articleID=17501499>.
6. Gallagher, Peter. Public InfoStructure – Inevitable Evolution: Unlocking Innovation for the Business of Government. Proc. of the Third Annual Open Source in Government Conference, George Washington University, Washington, D.C., 16 Mar. 2004 <www.egovos.org/Conferences/March_2004_Presentations>.
7. Perens, Bruce. The Open Source Initiative. Vers. 1.9. Open Source, 19 Oct. 2001 <www.opensource.org>.
8. Pavlicek, Russell. "Open Source Perspective: Open Source Origins." Processor 25.34 (22 Aug. 2003): 6 <www.processor.com/Editorial/article.asp?article=articles/p2534/06p34/06p34.asp>.
9. Raymond, Eric S. "The Cathedral and the Bazaar." Free-Soft.Org, 22 Nov. 1998 <www.free-soft.org/literature/papers/esr/cathedral-bazaar/cathedral-bazaar.html>.
10. United Nations. E-Commerce and Development Report 2003. United Nations Conference on Trade and Development, New York and Geneva, 2003: Chap: 4 "Free and Open Source Software: Implications for ICT Policy and Development." <www.unctad.org/en/docs/ecdr2003ch4_en.pdf>.
11. Green Hills Software. "Using Linux Software in Defense Systems Violates Every Principle of Security Says Green Hills Software's CEO and Founder." Santa Barbara, CA: Green Hills Software, 8 Apr. 2004 <www.ghs.com/news/20040408_AFEI.html>.
12. Groklaw. "CEO's of LynuxWorks and FSMLabs Reply to Green Hills' FUD." Groklaw. Ed. Pamela Jones. 11 Apr. 2004 <www.groklaw.net/article.php?story=20040411073918151>.
13. Singh, Inder. "LynuxWorks CEO, Dr. Inder Singh, Challenges Misrepresentative Claims Regarding Security in the Military." San Jose, CA: LynuxWorks, 2004 <www.lynuxworks.com/corporate/press/2004/linux-secure-military.php>.
14. Wheeler, David A. Open Source Software (OSS) and Security. Proc. of the Third Annual Open Source in Government Conference. George Washington University, Washington, D.C., 15-17 Mar. 2004 <www.egovos.org/Conferences/March_2004_

Presentations>.
15. Open Source Technology Group. Freshmeat.net 12 Sept. 2004 <http://freshmeat.net/about>.
16. Open Source Technology Group. Slashdot.org. Eds. Rob Malda, Jeff Bates, et al. 12 Sept. 2004 <http://slashdot.org/about.shtml>.
17. Kenwood, Carolyn A. "A Business Case Study of Open Source Software." Bedford, MA: The MITRE Corporation, July 2001 <www.mitre.org/work/tech_papers/tech_papers_01/kenwood_software/index.html>.
18. King, Julia. "A Sunny Forecast For Open Source." Computerworld 26 Apr. 2004 <www.computerworld.com/industrytopics/travel/story/0,10801,92583,00.html>.
19. Rapoza, Jim. "Can Open Source Provide Adequate Support?" eWeek 19 Apr. 2004 <www.eweek.com/article2/0,1759,1569380,00.asp>.
20. Dickerson, Chad. "CTO Connection: Open Source for a Song." InfoWorld 15 Aug. 2003 <www.info world.com/article/03/08/15/32OPconnection_1.html>.
21. Wacha, Jason B. "Open Source, Free Software, and the General Public License." Computer and Internet Law 20.3 (1 Mar. 2003): 20+.
22. Boos, Paul M. Using and Contributing to the Open Source Community While Supporting the Government. Proc. of the Third Annual Open Source in Government Conference. George Washington University, Washington, D.C., 15-17 Mar. 2004 <www.egovos.org/Conferences/March_2004_Presentations>.

Notes

1. In fact, the open source movement traces its origins (through the *free software* movement) to Richard Stallman's desire to enhance a proprietary printer driver [8].
2. The author of this article is not a lawyer. The information provided in this article is for informational purposes only and should not be construed as legal advice.

About the Author

David Tuma is the lead developer for the Software Process Dashboard Initiative, creating open source tools to support high-maturity software development processes. He first encountered open source software as a student at the Massachusetts Institute of Technology, and again later as a captain in the United States Air Force. As a strong supporter of open source, Tuma has been developing open source software on his own time for the past 10 years.

Software Process Dashboard Initiative
1645 E. HWY 193, STE 102
Layton, UT 84040-8525
Phone: (801) 771-4100
Fax: (801) 728-0595
E-mail: tuma@users.sourceforge.net

Web Sites

Open Source

www.opensource.org

The Open Source Initiative (OSI) is a non-profit corporation dedicated to managing and promoting the open source definition for the good of the community, specifically through the OSI Certified Open Source Software certification mark and program. You can read about successful software products and about OSI's certification mark and program on the Web site.

SourceForge.net

http://sourceforge.net

SourceForge.net is an open source software development Web site maintaining one of the largest repositories of open source code and applications available on the Internet. SourceForge.net provides free services to open source developers.

GNU Operating System – Free Software Foundation

www.gnu.org

The GNU [GNU's Not Unix] Project was launched in 1984 to develop a complete free software, Unix style operating system: GNU (pronounced guh-noo). The Free Software Foundation is the principal organizational sponsor of the GNU project.

National Technology Alliance

www.nta.org

The National Technology Alliance (NTA) is a U.S. government program established in 1987 to influence commercial and dual-use technology development with an emphasis on meeting national security and defense technology needs. The NTA's goal is to partner commercial technology solutions to government user technology needs and then create new or enhanced commercial products where the cost of development is leveraged across a broad user community.

Open Source Software Institute

http://oss-institute.org

The Open Source Software Institute is a non-profit organization comprised of corporate, government, and academic representatives whose mission is to promote the development and implementation of open source software solutions within U.S. federal and state government agencies and academic entities.

Samba

http://us4.samba.org

Samba is an open source/free software suite that provides seamless file and print services allowing for interoperability between Linux/Unix servers and Windows-based servers. Samba is freely available under the GNU General Public License. Samba is software that can be run on a platform other than Microsoft Windows such as Unix, Linux, IBM System 390, OpenVMS, etc.

freshmeat

http://freshmeat.net

freshmeat maintains one of the Web's largest index of Unix and cross-platform software, themes and related eye-candy, and Palm OS software. Thousands of applications and links to new applications are added daily. Each entry provides a description of the software, links to download it and obtain more information, and a history of the project's releases so readers can keep up-to-date on the latest developments.